Објаснување на Design Patterns кои се користени во апликацијата

Вовед

Ова е подетален документ кој објаснува како се имплементирани различните дизајн шаблони во ASP.NET MVC апликацијата за **Macedonian Stock Exchange (MSE)** податоци. Овие шаблони помагаат во организирањето, одржувањето и проширувањето на апликацијата.

Шаблони кои се користени:

- MVC
- Singleton
- Repository pattern
- DI

1. Model-View-Controller (MVC) архитектура

1.1 Model (Модел)

Моделот претставува апстракција на податоците од доменот. Во оваа апликација, тоа вклучува:

- **Класата `MseData`:**
  Оваа класа ги претставува податоците за акциите добиени од API услугата. Секое поле во класата кореспондира со атрибут од JSON одговорот.

  Код:

```
public class MseData
{
    public string Code { get; set; }
    public DateTime Date { get; set; }
    public decimal Minimum { get; set; }
    public decimal Maximum { get; set; }
    public decimal ClosePrice { get; set; }
    public decimal Volume { get; set; }
}
```

- **Ентитетски модел:**
  Класата `DataMseMVCContext` е креирана преку Entity Framework за да се поврзе со базата на податоци.

  Пример:

```
public class DataMseMVCContext : DbContext
{
    public DbSet<MseData> MseData { get; set; }
    public DbSet<DateClosePrice> DateClosePrices { get; set; }
}
```

1.2 View (Поглед)

Погледите ја презентираат информацијата до корисникот. Се користи Razor синтакса за прикажување на HTML во комбинација со податоци од моделите.

- **Пример:**
  Погледот `Data.cshtml` ја прикажува табелата со податоци за акциите и имплементира пагинација.

  Код:

```
<table class="table">
    <thead>
        <tr>
            <th>Code</th>
            <th>Date</th>
            <th>Minimum</th>
            <th>Maximum</th>
            <th>Close Price</th>
            <th>Volume</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.Code</td>
                <td>@item.Date.ToShortDateString()</td>
                <td>@item.Minimum</td>
                <td>@item.Maximum</td>
                <td>@item.ClosePrice</td>
                <td>@item.Volume</td>
            </tr>
        }
    </tbody>
</table>
```

- **Пагинација:**
  HTML елементи како `Previous` и `Next` се контролираат преку `ViewBag.PageIndex`.

1.3 Controller (Контролер)

Контролерот е посредник кој ги обработува барањата од корисникот, комуницира со моделот и враќа податоци до погледот.

- **Пример:** Метод за преземање на податоци:

```
public async Task<IActionResult> AllData(int pageIndex = 1)
{
    var response = await
_httpClient.GetAsync($"{_apiBaseUrl}/api/MseData");
    response.EnsureSuccessStatusCode();

    var content = await response.Content.ReadAsStringAsync();
    var data = JsonConvert.DeserializeObject<List<MseData>>(content);

    int pageSize = 10;
    var pagedData = data.Skip((pageIndex - 1) *
pageSize).Take(pageSize).ToList();

    ViewBag.PageIndex = pageIndex;
    ViewBag.TotalPages = (int)Math.Ceiling(data.Count /
(double)pageSize);

    return View("Data", pagedData);
}
```

2. Dependency Injection (DI)

DI се користи за да се инјектираат потребните зависимости (објекти) во контролерите и класите. Во апликацијата, `HttpClient` се инјектира преку `IHttpClientFactory`.

- **Конфигурација:**
  Во `Startup.cs`, се регистрираат сервисите:

```
services.AddHttpClient();
services.AddDbContext<DataMseMVCContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnecti
on")));
```

- **Пример:** Конструктор на контролер:

```
public MseDataController(IHttpClientFactory httpClientFactory)
{
    _httpClient = httpClientFactory.CreateClient();
    _apiBaseUrl = "https://localhost:7295";
}
```

3. Repository Pattern

Repository Pattern овозможува централизирање на логиката за пристап до базата, наместо да се користат директни повици до `DbContext` во контролерите.

- **Пример: Репозитори за податоци:**

```
public class MseDataRepository
{
    private readonly DataMseMVCContext _context;

    public MseDataRepository(DataMseMVCContext context)
    {
        _context = context;
    }

    public List<MseData> GetPagedData(int pageIndex, int pageSize)
    {
        return _context.MseData
                       .OrderBy(d => d.Date)
                       .Skip((pageIndex - 1) * pageSize)
                       .Take(pageSize)
                       .ToList();
    }
}
```

4. Asynchronous Programming

Асинхроните методи овозможуваат подобрување на перформансите со избегнување на блокирање на главната нишка додека се чека резултат од API или база.

- **Пример:**
  Метод кој презема листа на кодови:

```
private async Task<List<string>> GetCodes()
{
    var response = await
_httpClient.GetAsync($"{_apiBaseUrl}/api/MseData/GetCodes");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.ReadAsStringAsync();
    return JsonConvert.DeserializeObject<List<string>>(content);
}
```

5. Singleton Pattern

`HttpClient` се користи како **singleton**, што значи дека една инстанца се користи за сите HTTP повици. Ова е постигнато преку `IHttpClientFactory`.

- **Предности:**
  - Спречува **socket exhaustion**.
  - Го намалува времето на иницијализација.

Заклучок

Со примената на овие дизајн шаблони, апликацијата е структурирана и подготвена за понатамошно проширување. **MVC** овозможува јасна поделба на логиката, **Repository Pattern** ја намалува дупликацијата на кодот за базата, а **Dependency Injection** ја подобрува тестабилноста и управувањето со зависности.